Approfondimenti tematici

PuntoInformatico
Internet dal 1996
Libri

iPhone
la tua prima app

Come creare
facilmente un'app
per iPhone
e pubblicarla
su iTunes

EDIZIONI MASTER

iPhone
la tua prima app

Apple ha aperto una nuova frontiera per la programmazione mobile.
L'avvento di iPhone e di Apple Store ha dato modo agli sviluppatori, anche ai più piccoli, di fare breccia nel mercato mondiale delle applicazioni per dispositivi mobile.
Questo approfondimento tematico è pensato per mettere lo sviluppatore in condizione di creare facilmente App per iPhone e pubblicarle su iTunes.
La prima parte del testo è una guida passo passo alla creazione di una web application completa per iPhone utilizzando il framework iWebKit.
La seconda parte, invece, è dedicata alla pubblicazione delle App su App Store, con una marcata attenzione al rispetto delle regole per non incorrere nello "stop Jobs" e per favorire le prospettive di business dello sviluppatore mobile.
Il testo si chiude con un esempio di progetto pratico che guida il lettore alla creazione di un'applicazione pronta per iTunes.

PuntoInformatico | Libri

IPHONE: WEBAPP ALLA PORTATA DI TUTTI

CREIAMO UNA WEB APPLICATION COMPLETA PER IPHONE IN POCHI MINUTI CON IL FRAMEWORK IWEBKIT. SCOPRIREMO IL MODO PIÙ SEMPLICE PER ENTRARE NEL MONDO DELLA PROGRAMMAZIONE DELLO SMARTPHONE PIÙ AMBITO...







REQUISITI

Conoscenze richieste
HTML, CSS

Software
iWebKit, Notepad, Browser Safari

Impegno

Tempo di realizzazione



Volete entrare nel mondo delle programmazione iPhone, ma non avete un Mac a disposizione? Niente paura! Da oggi potrete creare i vostri programmi per iPhone anche da Windows! Come? Semplice: potete facilmente creare una WebApp sfruttando uno tra i più semplici framework free disponibili in rete: iWebKit. Non dovrete imparare alcun nuovo linguaggio, come l'Objective-C, anzi, potrete tranquillamente sfruttare le vostra attuali conoscenze di programmazione. Unico requisito: qualche nozione di HTML e CSS. Quella che creeremo oggi è una WebApp che fungerà da vetrina per la presentazione dei prodotti di un'azienda. Il risultato, mostrato in Fig. 1, sarà un programma simile alle classiche applicazioni Navigation-Based tipiche di iPhone, dove l'utente finale può "navigare", tra le diverse categorie fino ad arrivare al prodotto desiderato.

LE WEBAPP

Di cosa stiamo parlando esattamente? Di Web Application, ovvero applicazioni studiate per girare all'interno di un browser, nel nostro caso quello dell'iPhone all'interno di *Safari Mobile*. Non dovrete avere installato l'SDK (Software Development Kit) di Apple su Mac OS, (anche se al suo interno è disponibile l'utilissimo tool Dashcode), o imparare un nuovo linguaggio o paradigma di programmazione. Quello che invece vi serve è semplicemente un editor di testo, va bene anche il semplice notepad e, come detto, qualche nozione di HTML e CSS.
Alla base di ogni WebApp, infatti, vi è da strutturare un buon layout grafico, poi, con un po' di Javascript ed eventuali altri linguaggi di scripting, potrete realizzare più o meno tutto quello che desiderate: solo la vostra fantasia potrà limitare le vostre creazioni! Ebbene, nella realizzazione di questo primo aspetto di una WebApp, (grafica e layout), avrete più possibilità: rimanere con Apple e utilizzare il suo Dashcode, o sfruttare uno dei tanti framework disponibili in giro per il web, che vengono in nostro aiuto per semplificarci la vita. *iWebKit* è forse il più semplice fra questi framework, e vi farà entrare nel mondo iPhone in men che non si dica. A detta dell'autore, Christopher Plieger,



Fig. 1: Una visione di insieme di come si presenterà il nostro progetto finito

non occorrono neanche le poche conoscenze richieste (HTML e CSS) in quanto è possibile, creare la versione ottimizzata del proprio sito o della propria WebApp per iPhone sfruttando il Copia e Incolla.

CONTENUTO DEL FRAMEWORK

Prima di addentrarci nel lato pratico dell'articolo, analizziamo il contenuto del framework. Una volta scaricato il file *iWebKit5.03.zip* dal sito ufficiale *http://iwebkit.net/downloads* (in versione 5.03 al momento della stampa), ecco cosa troviamo all'interno del file *iWebKit 5.03.zip*:

- *changelog.pdf* con all'interno le modifiche effettuate al framework nelle varie versioni, dalla prima a quella attuale
- *Userguide.pdf* una guida passo passo (in inglese) con le istruzioni sull'uso del framework
- *license.pdf* la licenza d'uso del framework, (ricordiamo che stiamo parlando di uno strumento free)
- Una cartella *iWebKit* demo, una cartella contenente una demo pronta all'uso contenente la maggior parte dei componenti del framework (troverete un demo online da provare sul vostro iPhone anche sul sito ufficiale menzionato sopra)
- Una cartella *Framework*, contenente il vero "motore" che sfrutteremo in questo articolo.

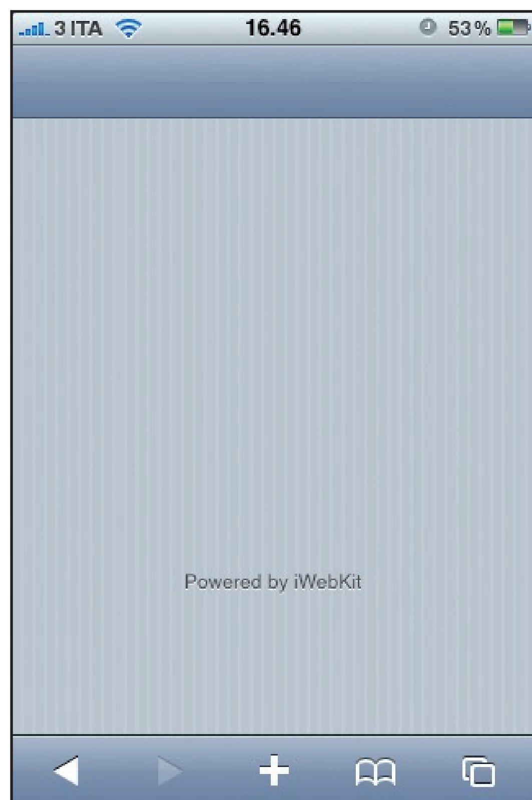


Fig. 2: La WebApp vuota e pronta alla personalizzazione che costituirà la nostra WepApp

Dentro quest'ultima cartella troviamo ulteriori quattro cartelle e un file HTML, *index.html*, potrà essere usato come punto di partenza per qualsiasi WebApp. Tra le quattro cartelle troviamo quella contenente il CSS che definisce l'aspetto della nostra WebApp, una cartella che contiene le funzioni in javascript, (al momento solo due, una per settare la vista in modalità fullscreen e l'altra per nascondere la *URLbar*, ovvero la barra dell'indirizzo di Safari Mobile). Infine troviamo ancora due cartelle, *images* e *thumbs*, contenenti gli elementi grafici da usare nelle nostre web application: pulsanti, switch e quant'altro possa tornarci utile. Facendo doppio clic sul file *index.html* si aprirà il browser che mostrerà una pagina contenente un semplice sfondo rigato e una top bar vuota (la classica NavigationBar delle applicazioni scritte in Objective-C).
Questo file, come detto, sarà il punto di partenza di ogni nostra WebApp sviluppata con questo framework. Ovviamente potrete ignorare il file e partire da zero, anche se non fareste altro che allungare inutilmente il vostro lavoro in quanto, il file *index.html*, non è nient'altro che un "modello" già preparato e collegato al css di *iWebKit*.

CREIAMO LA WEBAPP

Passiamo ora all'aspetto pratico e iniziamo a creare la nostra WebApp. Apriamo con un editor di testo il file *index.html* e analizziamolo. Come potete vedere si tratta di una semplice pagina HTML con la seguente struttura:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <link href="css/style.css" rel="stylesheet"
                    media="screen" type="text/css" />
    <script src="javascript/functions.js"
                        type="text/javascript"></script>
  </head>
  <body>
    <div id="topbar">
     </div>
      <div id="content">
      </div>
      <div id="footer">
     </div>
  </body>
</html>
```

Abbiamo volutamente omesso dal listato tutto quello che non serve ai fini di questa analisi, in





DISPLAY

Una delle caratteristiche da tenere sempre a mente durante lo sviluppo per un dispositivo mobile è la dimensione del display. Non avrete infatti molto spazio disponibile su cui "montare" i vari elementi della vostra applicazione, dovrete infatti accontentarvi, nel caso dell'iPhone, di una risoluzione, a schermo pieno, di 320x460 pixel. L'aspetto positivo è che conosciamo la dimensione esatta, certo è che non dovrete mai sottovalutare uno studio accurato dell'interfaccia utente finale per la vostra WebApp.



modo da visualizzare chiaramente la tipica struttura HTML del file.
All'interno dei tag *head* potete vedere i riferimenti al file *style.css* e al file contenente le funzioni in javascript *function.js*.
Dentro il body, invece, troviamo il contenuto vero e proprio della WebApp di Fig.2, nello specifico possiamo vedere le tre aree che andremo successivamente a personalizzare: *topbar*, *content* e *footer*.

LA TOPBAR

Iniziamo quindi col il primo elemento: la topbar, ovvero l'equivalente della navigation bar delle applicazioni classiche per iPhone. Dobbiamo innanzitutto scegliere un colore tra quelli disponibili (azzurra, nera o trasparente), specificare un titolo e aggiungere due pulsanti, uno a forma di freccia con raffigurata l'icona di una casa per tornare alla "*Home*" e che posizioneremo sulla sinistra e l'altro, un semplice tasto posizionato sulla destra e colorato in blu, che utilizzeremo per mostrare la pagina "*Info*". Arriveremo a ottenere quanto mostrato in Fig. 3:



Fig. 3: ***La topbar della nostra applicazione***

Per ottenere questo risultato dobbiamo modificare la porzione di codice dedicata alla topbar:

```
<div id="topbar"></div>
```

Possiamo personalizzarne il colore aggiungendo il parametro *class* a cui daremo il valore "*black*" per ottenere la topbar nera, e "*transparent*" per la versione trasparente. Per aggiungere il titolo "*io Programmo*" dovremo, invece, inserire all'interno della topbar il div "*title*" in questo modo:

```
<div id="topbar" class="transparent">
  <div id="title">ioProgrammo</div>
</div>
```

Per inserire i pulsanti, il procedimento è simile: dovremo quindi inserire il codice sempre all'interno del div *topbar*. Useremo *leftnav* per il pulsante a freccia e *bluerightbutton* per quello blu di destra:

```
<div id="topbar" class="transparent">
  <div id="title">ioProgrammo</div>
    <div id="leftnav">
      <a href="index.html"><img alt="home"
                        src="images/home.png" /></a>
  </div>
  <div id="bluerightbutton">
    <a href="info.html">INFO</a>
  </div>
</div>
```

Come vedete, si tratta di semplice HTML e, per l'inserimento dell'icona nel pulsante di sinistra, ce la siamo cavata con il classico tag *img*. Potete intuire con quanta semplicità potrete andare a personalizzare ogni aspetto delle vostre WebApp, il tutto senza dover studiare e capire chilometri di codice che spesso contraddistinguono framework proprietari.
Eccovi, per comodità, un elenco dei tipi di pulsanti disponibili nel framework, che potrete inserire a piacimento nelle topbar delle vostre WebApp:

- ***leftnav*** (pulsante a freccia sinistro)
- ***rightnav*** (pulsante a freccia destro)
- ***leftbutton*** (pulsante classico grigio sinistro)
- ***rightbutton*** (pulsante classico grigio destro)
- ***blueleftbutton*** (pulsante classico blu sinistro)
- ***bluerightbutton*** (pulsante classico blu destro)

Eventualmente potrete anche nidificare più *leftnav* (o *rightnav*), facendo però attenzione ai limiti imposti dalle dimensioni del display dell'iPhone (320pixel di larghezza). Per farlo, dovrete semplicemente aggiungere altri tag *a* come segue:

```
<div id="leftnav"><a
                    href="pagina1.html">Pag.1</a><a
                href="pagina2.html">Pag.2</a></div>
```

Per quanto riguarda la nostra topbar il lavoro è terminato, sappiate però che esistono altri elementi che possono essere inseriti al suo interno, o subito sotto. Stiamo parlando dei duo/trio buttons, che potete vedere in Fig. 4:

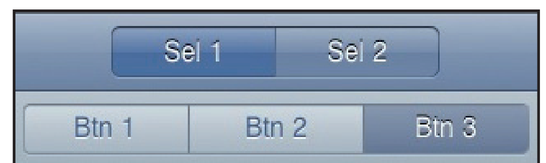


Fig. 4: ***In figura altri elementi che si possono inserire nella "zona" topbar***

DUO/TRIO BUTTONS

Se aveste necessità di utilizzare questi elementi sulla vostra *topbar* (facendo attenzione a non mettere il *title*, che verrebbe nascosto) dovrete



ORIENTAMENTO

Per chi non lo sapesse, l'iPhone è dotato di accelerometro integrato. Questo utilissimo elemento hardware offre all'utente la possibilità di variare l'orientamento del display a suo piacimento. Chiunque potrà di fatto scegliere se visualizzare le vostre creazioni in modalità portrait (schermata in verticale) o landscape (schermata in orizzontale). Dovrete sfruttare al meglio questa caratteristica, da tener ben presente durante la progettazione delle interfacce utente, se non volete ottenere effetti indesiderati.

utilizzare *duoselectionbuttons* o *triselectionbuttons*, ovviamente inserendo il codice all'interno del div *topbar*, se, invece, li volete subito sotto la barra di navigazione, vi basterà usare i *div duobutton* e *tributton*, questa volta subito dopo la chiusura della topbar, cosi come in questo esempio:

```
<div id="topbar">
    <div id="duoselectionbuttons"><a id="pressed"
      href="s1.html">Sel 1</a><a href="s2.html">Sel
                                          2</a></div>
</div>
<div id="tributton">
    <div class="links"><a href="b1.html">Btn
1</a><a href="b2.html">Btn 2</a><a id="pressed"
                              href="b3.html">Btn 3</a>
</div>
```

Potete infine usare *id="pressed"* nel tag *a* per visualizzare il tasto premuto dove occorre.
Prima di passare alla personalizzazione del div *content* vediamo un altro elemento che è possibile inserire dopo la chiusura della *topbar* e prima dell'apertura del *content*, si tratta di *doublead*, ovvero due box contenenti altrettanti file grafici, in stile App Store per intenderci (vedi Fig.5 subito sotto la topbar). Eccovi il codice:

```
<div id="doublead">
    <a href="http://www.ioprogrammo.it"
                              style="background-image:
                  url('pics/ioProgrammo.png')"></a><a
      href="http://www.devapp.it" style="background-
                   image: url('pics/devAPP.png')"></a>
</div>
```

PERSONALIZZIAMO IL CONTENT

Passiamo ora ad occuparci del content della nostra WebApp. Qui, gli elementi che possiamo introdurre, sono davvero molti e tutti, ovviamente, nel classico stile iPhone. Quello che andremo a costruire è un elenco contenente le categorie delle pubblicazioni Edizioni Master, come mostrato in Fig.5, successivamente, al clic (o meglio al "tap") su di una voce da parte dell'utente iPhone, si aprirà un elenco (che vedremo più avanti) contenente tutti i magazine Edizioni Master di quella categoria.
Gli elementi che andremo ad inserire da qui in avanti sono quelli che vedete sotto il doublead della Fig. 5, ovvero un titolo in grigio seguito da una casella di testo e successivamente un elenco graficamente molto simile alle classiche tabelle (*UITable*), con al loro interno più o meno dettagli, tipiche dell'iPhone.



Fig. 5: Aspetto della index completamente popolata

La maggior parte di questi elementi vengono gestiti con i tag *ul* e *li*, usati in HTML per gestire gli elenchi. Sarà poi grazie al parametro *class* del tag *li* che visualuzzeremo una voce della tabella (*menu*) piuttosto che una casella di testo (*textbox*). Inseriamo il seguente codice all'interno del *div content*:

```
<span class="graytitle">Edizioni Master</span>
<ul class="pageitem">
  <li class="textbox">
    <span class="header">Magazines</span>
    <p>Di seguito i Magazines Edizioni Master
                        disponibili in tutte le edicole.</p>
  </li>
</ul>
<span class="graytitle">Categorie</span>
<ul class="pageitem">
  <li class="menu">
    <a href="digitechProfessional.html"
      <span class="name">DIGITECH
                              PROFESSIONAL</span>
      <span class="comment">Vai</span>
      <span class="arrow"></span>
    </a>
  </li>
  <li class="menu">
    <a href="demo.html">
      <span class="name">HOME
                             ENTERTAINMENT</span>
```





NOTA

STATUS BAR

Se volete cambiare il colore della Status Bar, ovvero quella barra orizzontale alta 20 pixel, colorata normalmente in grigio, in cui viene visualizzato l'orologio o la carica della batteria negli iPhone, non dovete far altro che aggiungere, all'interno del tag head, una riga di codice. Potrete lasciare la Status Bar inalterata non aggiungendo nulla oppure modificarla da grigia a nera, o nera in trasparenza. Eccovi il codice:
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
<meta name="apple-mobile-web-app-status-bar-style" content="black-translucent"/>





NOTA

VIDEO YOUTUBE

Nell'articolo non lo abbiamo accennato, ma potete interagire anche con l'applicazione nativa YouTube, presente in ogni iPhone e iPod Touch. Il funzionamento è davvero semplice, basterà infatti copiare semplicemente l'URL del video desiderato ed inserirlo nella seguente riga di codice che utilizzerete poi dove meglio credete all'interno della vostra WebApp:
*Guarda *
Si tratta del semplice tag usato in HTML per l'aperture dei link: a.

```
      <span class="comment">Vai</span>
      <span class="arrow"></span>
    </a>
  </li>
  <li class="menu">
    <a href="demo.html">
      <span class="name">DIGITECH END
                              USER</span>
      <span class="comment">Vai</span>
      <span class="arrow"></span>
    </a>
  </li>
  <li class="menu">
    <a href="demo.html">
      <span class="name">VIDEOGAMES &amp;
                              CARTOON</span>
      <span class="comment">Vai</span>
      <span class="arrow"></span>
    </a>
  </li>
  <li class="menu">
    <a href="demo.html">
      <span class="name">LIFESTYLE</span>
      <span class="comment">Vai</span>
      <span class="arrow"></span>
    </a>
  </li>
</ul>
```

Abbiamo definito due sezioni distinte, ognuna racchiusa tra tag *ul* e precedute da un titolo. Unica differenza è che il parametro *class* del tag *li* avrà valore *textbox* nel caso in cui stiamo inserendo la casella di testo e *menu* per gli elementi della tabella (l'elenco). Per ogni riga della tabella, inoltre, possiamo definire un nome (*name*), ovvero il testo visualizzato, un commento (*comment*), che verrà visualizzato in grigio sulla destra ed eventualmente mostrare una freccia (*arrow*) che indica all'utente che può proseguire nella navigazione nella WebApp cliccando semplicemente su di una cella. Ricordiamo che stiamo sempre parlando di HTML, quindi potrete personalizzare e modificare come meglio credete quanto mostrato fino ad ora. Ad esempio, per omettere un commento o la freccia, non dovrete far altro che togliere quella riga di codice.



Fig. 6: Screenshot dopo il click su "Chiamaci", "Scrivici" e "Vieni a trovarci"

INTERAGIRE CON LE APP NATIVE

Concludiamo la personalizzazione del *content* della pagina *index.html* per la nostra WebApp, andando a inserire un'ulteriore elenco (sempre in stile *UITable*) sotto l'elenco categorie. Creeremo questa volta una sezione dedicata ai contatti, che sfrutterà qualche funzione in più rispetto a quanto visto fino ad ora. Saremo infatti in grado di interfacciarci con le app native *telefono*, *mail* e *mappe* di iPhone, passando ad esse, al "tap" sulla rispettiva voce, alcuni parametri, come il numero di telefono da chiamare, gli indirizzi e-mail a cui scrivere, eventuale oggetto e corpo della mail o un link di Google Maps che ci permetterà di aprire l'applicazione mappe posizionato nella luogo desiderato. Per far ciò aggiungiamo il seguente codice prima della chiusura del *div content*:

```
<span class="graytitle">Contatti</span>
<ul class="pageitem">
  <li class="menu">
    <a class="noeffect" href="tel:+3909848319200">
      <img alt="Description"
                    src="thumbs/telephone.png" />
      <span class="name">Chiamaci</span>
      <span class="comment"></span>
      <span class="arrow"></span>
    </a>
  </li>
  <li class="menu">
    <a class="noeffect"
 href="mailto:ioprogrammo@edmaster.it?cc=info@dev
          app.it&subject=Richiesta Informazioni
 WebApp&body=Alla C.A. redazione ioProgrammo.it">
      <img alt="Description" src="thumbs/mail.png"
                                              />
      <span class="name">Scrivici</span>
      <span class="comment"></span>
      <span class="arrow"></span>
    </a>
  </li>
  <li class="menu">
    <a class="noeffect"
 href="http://maps.google.it/maps?f=q&source=s_q&
          hl=it&geocode=&q=Via+Ariberto,+24+-
+20123+Milano&sll=41.442726,12.392578&sspn=18.
100779,37.749023&ie=UTF8&hq=&hnear=Via+Ariber
      to,+24,+20123+Milano,+Lombardia&z=16">
```

```
    <img alt="Description" src="thumbs/maps.png" />
    <span class="name">Vieni a trovarci</span>
    <span class="comment"></span>
    <span class="arrow"></span>
   </a>
  </li>
</ul>
```

Anche in questo caso il codice è molto semplice. Grazie a questa riga di codice:

```
<span class="graytitle">Contatti</span>
```

verrà aggiunto un titolo grigio alla nuova sezione della pagina. Ai successivi elementi dell'elenco, invece, molto simili a quelli della prima parte del *content*, abbiamo semplicemente aggiunto un link (con all'interno i parametri da passare alle app native) e un'icona pressa dalla cartella thumbs inclusa nel framework. Con il *content* del file *index.html* abbiamo terminato, per quanto riguarda il footer, lo lasciamo così com'è, anche se potremmo personalizzarlo a piacere seguendo le stesse regole mostrate fino ad ora.

ELENCO IN STILE ITUNES

Vediamo ora come creare un elenco simile a quello che trovate nell'iTunes Store (Fig. 7), con tanto di stellette di valutazione per ogni



Fig. 7: Elenco in stile App Store su iTunes

elemento. Facciamo una copia del file *index.html* appena creato e rinominiamolo in *digitechProfessional.html*. Apriamo il file con un editor di testo e cancelliamo tutto il codice all'interno del div *content*. Il resto, per comodità, lasciamolo inalterato. Se visualizziamo un'anteprima nel browser del file (ricordate che iWebKit è studiato per lavorare con Safari), vedrete che abbiamo mantenuto la *topbar* con tutti i suoi elementi incluso l'elemento subito sotto la *doublead*.
Nel framework iWebKit esistono diversi elementi per la visualizzazione degli elenchi e nello specifico:

- elenco classico
- elenco in stile App Store
- elenco in stile iTunes Store
- elenco in stile libreria iPod

Noi arriveremo al risultato mostrato in Fig. 7 tramite un elenco simile a quelli visti in precedenza ricavati grazie a *class="pageitem"* nel tag *ul*. Nel tag *li*, invece, imposteremo il valore di *class* a *store* (anziché *textbox* o *menu* visti in precedenza)
Inseriamo quindi, subito dopo l'apertura del *content,* un titolo:

```
<span class="graytitle">Digitech Professional</span>
```

e continuiamo inserendo il tag *ul* all'interno del quale inseriremo poi i vari elementi dell'elenco:

```
<ul class="pageitem">
</ul>
```

All'interno di *ul* inseriremo, quindi, tutti gli elementi che desideriamo in questa forma:

```
<li class="store">
  <a href="ioprogrammo.html">
    <span class="image" style="background-image:url('pics/ioProgrammo.jpg')"></span>
    <span class="comment">La rivista di software development più venduta in Italia</span>

    <span class="name">ioProgrammo</span>
    <span class="stars5"></span>
    <span class="starcomment">€ 9,99 Vers. PLUS</span>
    <span class="arrow"></span>
  </a>
</li>
```

Anche qui la struttura è molto semplice e potremo personalizzarla a nostro piacimento.





INSTALLAZIONE

Per installare e testare la vostra WebApp, la soluzione più semplice è quella di fare l'upload dell'intero progetto nello spazio dedicato che avete nel vostro eventuale piano di Hosting associato al vostro dominio, Potete poi raggiungere l'applicazione semplicemente inserendo l'indirizzo web in Safari Mobile, proprio come fareste per raggiungere un sito web. Se non possedete un Hosting, potete sempre usufruire di uno degli innumerevoli servizi gratuiti disponibili in Rete, come quello offerto da alterVISTA (http://it.altervista.org/)



Potremmo inserire un'immagine per ogni elemento, questa deve essere di dimensioni 90x90pixel, nel caso fosse più grande, verrà ridimensionata in automatico. Fate attenzione che se le immagini utilizzate non sono in scala, verranno "distorte". Dopo l'immagine, troviamo un commento che verrà posizionato in alto, subito sotto il nome e a seguire possiamo aggiungere le stelle di valutazione grazie all'istruzione

```
<span class="stars5"></span>
```

Variando il numero di fianco *stars*, potremo visualizzare da 0 a 5 stelle. Possiamo inoltre aggiungere un commento subito dopo le stelle, che noi abbiamo utilizzato per indicare il prezzo della rivista. Infine, l'ultima riga, non fa altro che inserire la freccia a destra per scendere ancora di un livello nella navigazione, in parole povere, per visualizzare il dettaglio della rivista. Potrete inserire tutti gli elementi che volete mantenendo semplicemente questa struttura, volendo, potrete inoltre personalizzare il tutto a piacere rimuovendo o aggiungendo i vari componenti.



L'AUTORE

Rino Picardi, System Administrator di professione e appassionato di informatica e tecnologie da sempre, è il fondatore della community italiana di Programmazione per iPhone devAPP (http://www.devapp.it). Con le sue pubblicazioni e quelle dei suoi collaboratori, guida tutti gli appassionati della "mela" ad addentrarsi nell'affascinante mondo della programmazione mobile, dai primi passi fino alla pubblicazione in App Store. Potete contattarlo al seguente indirizzo e-mail: info@devapp.it

RIFINITURE

Seguendo lo stesso metodo potrete creare tutte le pagine che volete e organizzarle come meglio credete, proprio come se fosse un sito web. Nel nostro progetto potete vedere altre pagine HTML create tutte con lo stesso meccanismo: *demo.html*, *ioProgrammo.html* e *info.html*.

Queste contengono tutti gli elementi visti fino ad ora. Potrete inoltre spulciare la guida all'uso inclusa nel kit del framework per introdurre eventualmente altri elementi: il funzionamento rimarrà pressoché invariato. Prima di concludere vorremmo proporvi ancora tre aspetti utili per ogni WebApp, ovvero la visualizzazione in modalità *Full Screen*, la *Splashscreen* (la vista che viene mostrata durante il caricamento della WebApp) e la personalizzazione dell'icona che comparirà nella springboard del vostro iPhone. Per quanto riguarda la modalità di visualizzazione *Full Screen* abbiamo poco da fare, essendo un accorgimento lato utente, che, premendo il tasto "+" in Safari Mobile, potrà aggiungere la WebApp nell'elenco delle proprie applicazioni. Proprio per questo motivo sarebbe carino prevedere una Splashscreen e un'icona studiate appositamente per la vostra WebApp. Per inserire la *Splashscreen* ci basterà inserire nel progetto, ad esempio nella cartella *pics*, un'immagine in formato *.png* grande 320x460 pixel, che chiameremo *startup.png*. Dovremo poi inserire il seguente codice prima della chiusura del tag *head*:

```
<link href="pics/startup.png" rel="apple-touch-
                              startup-image" />
```

Per l'icona il procedimento è analogo, creiamo la stessa in formato *.png* di dimensioni 58x58pixel, che chiameremo, ad esempio, image.png, e la inseriamo sempre nella cartella pics. Anche in questo caso occorrerà inserire, sempre prima della chiusura dell'head, il seguente codice:

```
<link rel="apple-touch-icon"
                    href="pics/image.png"/>
```

L'avvio della splash, così come il salvataggio dell'icona desiderata, sarà del tutto automatica. Potete vedere un'esempio di Splashscreen in Fig. 8. In conclusione possiamo affermare che iWebKit è la soluzione ideale per entrare nel modo dell'iPhone nel minor tempo possibile. Pochi i requisiti necessari per imparare a maneggiare la collezione di oggetti disponibili gratuitamente e alla portata di tutti al suo interno.

Rino Picardi



Fig. 8: Un piccolo "capolavoro" disegnato nel nostro fidato paint...

LA NOSTRA APP È SULL'APPLE STORE!

DOPO TANTI MESI DEDICATI ALLA REALIZZAZIONE DELLE APP PER IPHONE, È TEMPO DI SCOPRIRE COME PUBBLICALE SU APPLE STORE, RISPETTANDO LE REGOLE DI APPLE E SEGUENDO TUTTE LE PROCEDURE NECESSARIE PER NON INCORRERE NELLO STOP JOBS!







Conoscenze richieste
Objective-C

Software
Mac OS 10.5.x o superiore (10.6.x per l'SDK 3.x)

Impegno



Tempo di realizzazione



Dopo i numerosi articoli che sono stati presentati su questa rivista e in cui abbiamo mostrato alcuni aspetti fondamentali della programmazione su iPhone/iPod Touch, apriamo una parentesi dedicata alla pubblicazione su Apple Store. In queste pagine illustreremo tutte le procedure necessarie affinché la vostra applicazione venga approvata, cercando di minimizzare gli errori più frequenti allo scopo di evitare possibili rifiuti da parte del team di valutazione di Apple che, come sapete, ha riservato per sé un diritto di veto "quasi" inappellabile.

IL DECALOGO DA SEGUIRE PRIMA DELLA PUBBLICAZIONE

Prima di inviare un'applicazione su Apple Store è necessario seguire i seguenti passi:

1) Acquisto di una licenza annuale come sviluppatore;
2) Verifica che l'applicativo rispecchi pienamente le regole dettate nel documento chiamato iPhone/iPad Human Interface Guidelines;
3) Verifica che l'applicativo non utilizzi API non pubbliche e framework non ufficiali;
4) Creazione di un sistema di chiavi necessarie per firmare il proprio applicativo sul Provisioning Portal;
5) Creazione del profilo di pubblicazione in XCode;
6) Creazione di un file di Entitlements;
7) Modifica delle opzioni di compilazione del progetto;
8) Compilazione e packaging;
9) Pubblicazione su iTunes Connect;
10) Invocazione della protezione degli dei, affinché la nostra app riscontri il successo che ci aspettiamo.

Quasi tutti i passi devono essere ripetuti per ogni singola applicazione, a parte ovviamente l'acquisto della licenza. Tratteremo in dettaglio ognuno degli step appena descritti, iniziando proprio con l'acquisto della licenza di sviluppatore.

L'ACQUISTO DELLA LICENZA

Se per testare le proprie applicazioni è sufficiente adoperare l'SDK fornito gratuitamente sul sito Apple (http://developer.apple.com/iphone), ciò non è più sufficiente quando si decide di pubblicare su Apple Store, per tale motivo è necessario acquistare una licenza annuale del costo di 79 euro, con una carta di credito (vanno bene anche quelle "prepagate" oltre a quelle "normali"). Apriamo una parentesi sui tipi di licenza disponibili: ne esistono numerosi, ma i due più utilizzati sono lo "*Standard Program*" e l'"*Enterprise Program*", a parte la differenza di prezzo (circa 200$ in più per quest'ultimo) il secondo, indirizzato a società con più di 500 dipendenti, inibisce la possibilità di pubblicare su Apple Store, consentendo invece di distribuire internamente ai propri dipendenti (in-*house distribution*) senza alcun vincolo i propri software per il dispositivo Apple; lo Standard Program è utilizzabile sia da singoli sviluppatori che da società che desiderano pubblicare su Apple Store. A parte la necessità di adoperarli per la pubblicazione, esistono altri vantaggi forniti da questi programmi, primo fra tutti la possibilità di testare i propri software su dispositivi fisici, iPhone, iPod o iPad, consentendo di realizzare un test più accurato sia sulle prestazioni che su come il sistema operativo reale interpreta e mostra i vostri contenuti. Un ulteriore pregio è quello di consentire la cosiddetta *Ad Hoc Distribution*, procedura che consente di inviare ad altri dispositivi il vostro software



Fig. 1: Selezionando Enroll Now si inizia la fase di registrazione

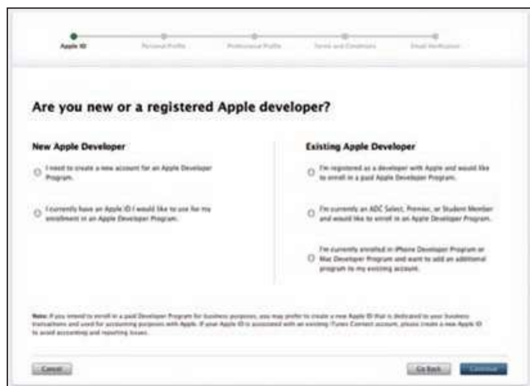


Fig. 2: Possiamo scegliere se adoperare un account preesistente o crearne uno nuovo

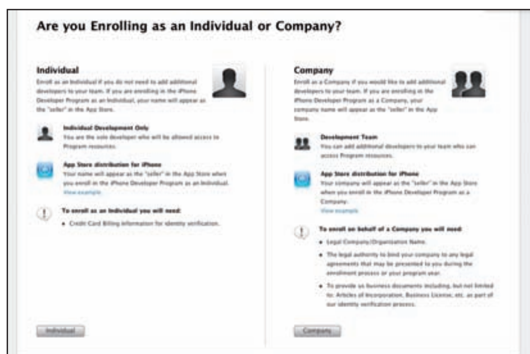


Fig. 3: In questa schermata si decide se registrarsi come società o come singolo sviluppatore

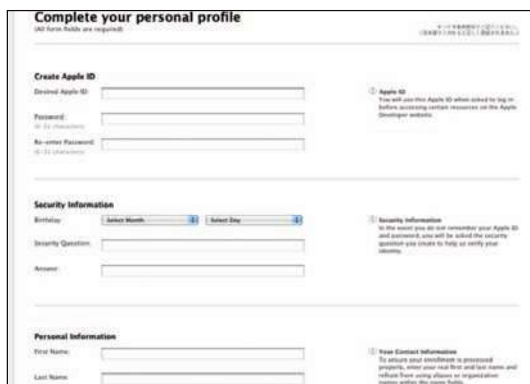


Fig. 4: Alcune delle informazioni che l'utente deve inserire per acquistare la licenza

per scopi di test o pubblicitari, ma attenzione: questo tipo di distribuzione perde la sua validità (scade) dopo alcuni mesi, obbligandovi automaticamente ad inviare nuovamente il software. Non è quindi un sistema di distribuzione definitivo, ma temporaneo, nato per consentire l'invio a utenti test che non fanno parte del team di sviluppatori. Con entrambi i programmi si acquisisce anche la possibilità di poter scaricare versioni pre-release, accedere al forum degli sviluppatori Apple, servizio utile per richiedere aiuto e ottenere suggerimenti da altri sviluppatori, e si ottiene una garanzia tecnica Apple per due anni. Per registrarsi come sviluppatore basta accedere all'indirizzo http://developer.apple.com/programs/iphone e cliccare sul bottone *Enroll Now*, a questo punto si dà inizio ad una serie di passi obbligatori che forniscono sufficienti informazioni ad Apple per registrarvi, ovviamente si dovrà scegliere *Standard Program* e, nel caso in cui non possediate un account Apple, è necessario crearne uno sempre all'interno di questa procedura. Purtroppo non possiamo mostrarvi tutte le immagini delle fasi di registrazione per le limitazioni imposte dall'iPhone Agreement, documento che si deve accettare obbligatoriamente, dopo la fase di inserimento dei propri dati personali. Al termine di tutti i vari passi, è possibile che Apple provi a contattarvi telefonicamente, quasi sempre da parte di operatori in lingua inglese, per verificare che abbiate inserito i dati correttamente, per le società questo passo è richiesto da Apple che, inoltre, pretende di ricevere un documento legale tramite un numero di fax che attesti l'effettiva esistenza della stessa dal punto di vista legale (iscrizione alla Camera di Commercio). Dopo alcuni giorni riceverete una serie di e-mail di conferma che vi consentiranno di finalizzare la registrazione. Al termine, avrete accesso ai vari portali necessari per pubblicare i vostri software: *iPhone Provisioning Portal* e *iTunes Connect* in primis. Il Provisioning Portal svolge la funzione di portale, è necessario accedervi per creare i certificati che vengono adoperati per testare su dispositivi fisici, e inviare le proprie applicazioni sull'Apple Store; iTunes Connect è il servizio che accetta i programmi realizzati per iPhone/iPod/iPad e rappresenta l'ultimo passo necessario per la pubblicazione.

LA VERIFICA DELL'APP CON LA HUMAN INTERFACE GUIDELINES

Prima di ipotizzare un qualunque invio ad





TRE LINK DA NON DIMENTICARE

http://developer.apple.com/programs/iphone presso questo indirizzo è possibile acquistare la licenza per divenire uno sviluppatore iPhone.

Con **iTunes Connect** gli sviluppatori possono pubblicare le loro applicazioni Connect: https://itunesconnect.apple.com

Con l'**iPhone Provisioning Portal** abbiamo accesso a tutti i certificati necessari alla pubblicazione dell'app: http://developer.apple.com/iphone/manage/overview/index.action





NOTA

GESTIONE DELL'ACCOUNT
Questo il link per la creazione dell'account, per scaricare l'SDK e consultare la documentazione previa registrazione gratuita: http://developer.apple.com/iphone/

Apple, è necessario effettuare preventivamente delle verifiche sul proprio software in modo da sincerarsi che rispecchi tutte le regole elencate nel documento chiamato: *Human Interface Guidelines* (*HIG* in alcuni documenti), disponibile sia nella versione iPhone che iPad. Il documento in questione, disponibile sia online in formato HTML che PDF, e nella documentazione allegata ad XCode, rappresenta il testo che detta le regole su cosa i valutatori Apple ritengono ammissibile in un'applicazione iPhone e cosa invece li obbligherà a rifiutarla. Negli articoli fin qui proposti abbiamo fatto attenzione a non adoperare nessun componente in modo non consentito, e abbiamo seguito il più possibile le informazioni contenute in tale testo. Non leggere almeno una volta questo documento disponibile al momento solo in lingua inglese, porterà generalmente ad una sequenza di rifiuti da parte di Apple, questo è dovuto al fatto che le informazioni contenute al suo interno rappresentano quei binari che ogni sviluppatore deve seguire per mantenere la propria applicazione coerente con tutte le altre già presenti su Apple Store. I problemi maggiori si riscontrano in genere quando:

- Il software presenta uno o più crash;
- Si personalizzano alcuni componenti visuali predefiniti rendendo il loro comportamento poco intuitivo graficamente oppure differente semanticamente da ciò che un utente si aspetterebbe;
- Si utilizzano icone il cui significato è fuorviante, oppure adoperando quelle fornite nell'SDK in contesti non consentiti;
- Se non si mostrano avvertimenti adoperando ad esempio gli *UIAlertView* di cui abbiamo trattato in precedenti articoli quando si presentano situazioni impreviste, quali mancanza o cadute di connessione ad Internet, o altri tipi di errori di cui l'utente deve essere informato;
- Non si verifica preventivamente che un dispositivo ha o no accesso ad un determinato componente hardware (come GPS, microfono, giroscopio, bussola) perché questo non è abilitato oppure non presente fisicamente, e non si notifica tale carenza all'utente;
- Si utilizzano icone diverse tra l'Apple Store e quella interna al software;
- Si supporta la rotazione dell'interfaccia senza riorganizzarne il layout.

La serie di errori possibili è talmente vasta che ne abbiamo presentato solo un ristretto sottoinsieme, comunque all'interno di questa guida per ogni singolo componente si trova una descrizione che ne identifica dettagliatamente lo scopo, le funzionalità, come dovrebbe e come non dovrebbe essere adoperato e customizzato. Trovano posto anche paragrafi dedicati a cosa l'utilizzatore di un certo servizio/componente grafico si aspetta. Anche se questo documento sembra rappresentare solo una lista di limitazioni alla propria creatività deve essere considerato una fonte di informazioni fondamentali. È comunque sempre possibile realizzare software interamente personalizzati senza adoperare alcun componente grafico predefinito, in tal caso si bypassano molti dei check effettuati dal team di approvazione Apple. Bisogna dire che le due Human Interface Guidelines non sono a volte l'unica motivazione che porta al rifiuto del proprio software ma in genere rappresentano il motivo principale per la quasi totalità dei reject. Questa guida cambia con una certa frequenza, viene corretta, e ampliata dai tecnici Apple anche in base alle novità introdotte nelle nuove versioni degli SDK rilasciati e conviene verificare con una certa frequenza , in genere mensile, se ha subito aggiornamenti. Il controllo adoperando questa guida avviene quindi con un approccio a "a scatola nera" non conoscendo il codice ma semplicemente interagendo con il vostro software e verificando come si presenta visivamente e come si comporta.

METODI O FUNZIONI NON PUBBLICHE

Un'altra motivazione, meno frequente della precedente ma abbastanza comune per chi è alle prime mani con lo sviluppo di un software, è quella dovuta al fatto che il proprio applicativo non rispetta quel dogma di Apple che impone di non adoperare funzioni, metodi, e classi non disponibili ufficialmente nell'SDK. Questa operazione di controllo viene effettuata da Apple per evitare alcuni problemi: si pensi al caso in cui queste funzionalità nascoste subiscano cambi, come avviene comunemente in una versione successiva dell'SDK, ciò porterebbe ad un comportamento diverso del proprio applicativo che in genere lo porta ad un crash. Bisogna capire che se queste funzionalità sono nascoste ci sono diversi motivi alla base, certo alcuni sono pienamente plausibili, altri meno, e spesso obbligano a trovare soluzioni meno efficaci: si pensi al piacevole effetto grafico adoperato dagli sviluppatori Apple per sfogliare gli iBook, questo non è disponibile pubblicamente, al momento è presente solo quello verticale, (*curlUP*, *curlDown* e parziale)

e le soluzioni alternative disponibili in rete non sono perfettamente speculari. Lo sviluppatore meno pratico in genere incappa in questo problema quando cerca una soluzione software per il proprio applicativo ed effettua il tipico copia ed incolla incondizionato da risorse disponibili online, in genere forum o blog. L'accesso a queste risorse nascoste è possibile perché il linguaggio Objective-C non consente di nascondere completamente funzioni e campi, così alcuni sviluppatori, adoperando interrogazioni tramite codici opportunamente realizzati, hanno avuto accesso ad una lista completa di tutte le informazioni pubbliche e non di ogni classe e struttura dati. Il compilatore invocato da XCode (GCC o LLVM) provvede ad avvisare sempre tramite un warning se si effettuano operazioni non consentire, il messaggio di avviso in genere contiene il messaggio "*could not <TIPO>*", *<TIPO>* potrebbe essere ad esempio "*respond to selector*", ad indicare che si invoca un metodo non esposto pubblicamente. Le regola di non adoperare API private è inoltre presente nell'*iPhone Developer Program License Agreement*, un documento che deve essere accettato quando si diviene sviluppatori iPhone e si acquista la licenza per pubblicare:

3.3.1 — Applications may only use Documented APIs in the manner prescribed by Apple and must not use or call any private APIs.

Questo estratto non lascia quindi spazio ad altre interpretazioni. Per qualunque dubbio basta controllare se nella documentazione ufficiale il campo, la proprietà, la funzione o il metodo adoperato sono presenti.

TECNOLOGIE NON UFFICIALI

Prima dell'annuncio ufficiale dell'iOS4, si erano aperti diversi spiragli per quegli sviluppatori che preferivano adoperare altri linguaggi invece dell'Objective-C, per motivi economici (non richiedono spesso l'acquisto di un computer Mac), o semplicemente perché preferivano utilizzare le conoscenze già possedute. I due attori più importanti sono stati ActionScript (utilizzando il nuovo Adobe Flash CS5 Professional) e .NET con MonoTouch, strumento basato su Mono. Apple ha deciso di impedire l'utilizzo di queste tecnologie sempre all'interno della sezione 3.3.1 dell'iPhone Developer Program License Agreement:

Applications must be originally written in Objective-C, C, C++, or JavaScript as executed by the iPhone OS WebKit engine, and only code written in C, C++, and Objective-C may compile and directly link against the Documented APIs (e.g., Applications that link to Documented APIs through an intermediary translation or compatibility layer or tool are prohibited).

Anche qui non ci sono tante vie da percorrere, C, C++, Objective-C o JavaScript sono i linguaggi che Apple richiede vengano utilizzati. È importante precisare che questa limitazione non impedisce di adoperare alcuna libreria disponibile in rete realizzate nei linguaggi suddetti, a patto ovviamente che non acceda a API non documentate. Il rischio, adoperando una tecnologia diversa da quella richiesta da Apple, potrebbe comportare la non approvazione dell'applicazione, rendendo vano tutto il lavoro svolto.
Il consiglio è quello di evitare vie alternative e impegnarsi nello studio dell'Objective-C, approfondendo in modo dettagliato ogni aspetto inerente questo potente linguaggio di programmazione.

VALIDATE BUILD PROJECT: UNA SICUREZZA IN PIÙ

Per ridurre al minimo le possibilità di un rifiuto in fase di invio è stato introdotta all'interno di XCode 3.2 una nuova funzionalità chiamata *Validate Build Product*, attivabile tramite un ckeckbox disponibile all'interno della configurazione di Build del progetto; selezionando tale campo si richiede che vengano effettuati molti dei check che i valutatori Apple effettuano quando ricevono un'applicazione, non è un controllo che garantisce che il proprio software sia esente da problemi, ma riduce al minimo tali eventualità. Un esempio di avvertimento che tale tool fornisce è quello dovuto ad un errore nelle dimensioni delle icone inserite nel progetto, che porterebbero ad un rifiuto automatico del software, ancora prima di una qualunque valutazione da parte dei tecnici Apple. Abilitare tale campo è quindi conveniente in qualunque progetto, anche se incrementa di qualche secondo il tempo necessario per la compilazione. Nel prossimo articolo mostreremo come creare i certificati, li adopereremo per firmare il nostro applicativo, pubblicheremo la nostra applicazione su Apple Store e adopereremo iTunes Connect per visualizzare i guadagni e i download ottenuti.

Andrea Leganza





Andrea Leganza, laureato in Ingegneria Informatica, certificato Adobe ACE - Adobe Flex 3 and AIR Certified Expert, EUCIP Core, Sun Certified Programmer for JAVA 6, istruttore di nuoto FIN di 2° Livello, è attualmente impegnato in numerosi progetti multimediali, anche con iPhone e iPad, con alcune società nazionali ed internazionali, è contattabile su neogene@tin.it o su www.leganza.it.

LA NOSTRA APP È SULL'APPLE STORE

PUBBLICARE UN'APPLICAZIONE SU APPLE STORE ADOPERANDO I PORTALI IPHONE PROVISIONING PORTAL E ITUNES CONNECT: SCOPRIAMO COME FARLO AL MEGLIO, SEGUENDO TUTTO IL PERCORSO CHE ARRIVA FINO ALLA PUBBLICAZIONE

SECONDA PARTE



REQUISITI

Conoscenze richieste
Objective-C

Software
Mac OS 10.5.x o superiore (10.6.x per l'SDK 3.x)

Impegno

Tempo di realizzazione

Nell'articolo precedente abbiamo spiegato le fasi iniziali necessarie per avviarsi verso il rilascio della propria applicazione su Apple Store, abbiamo infatti acquistato la licenza come singolo sviluppatore, per poi passare ad un controllo sul codice adoperando il documento chiamato HIG, Human Interface Guideline, che permette con le sue linee guida di ridurre al minimo i possibili rifiuti da parte del team di valutazione interno di Apple. In questa seconda ed ultima parte effettueremo gli ultimi passi che ci consentiranno di pubblicare su Apple Store.

LE OPERAZIONI DA SEGUIRE

I passi che sono stati descritti nell'articolo precedente sono i seguenti:

1. acquisto di una licenza annuale come sviluppatore
2. verifica che l'applicativo rispecchi le regole dettate nel documento chiamato *iPhone/iPad Human Interface Guidelines*
3. verifica che l'applicativo non utilizzi API non pubbliche e framework non ufficiali

Quelli che ancora dobbiamo completare sono:

1. creazione di un sistema di chiavi necessarie per firmare il proprio applicativo sul Provisioning Portal;
2. Creazione di un file di Entitlements in XCode;
3. Creazione della configurazione di pubblicazione (che chiameremo "*Production*") in XCode;
4. Modifica delle opzioni di compilazione del target del progetto;
5. Compilazione e packaging;
6. Pubblicazione su iTunes Connect

Purtroppo non ci è possibile mostrare le immagini delle fasi necessarie per la pubblicazione per le limitazioni imposte da Apple, prevalentemente per quanto riguarda il portale online, cercheremo quindi di descrivere i singoli passi con il massimo in dettaglio.

LA RICHIESTA DELLE CHIAVI

Firmare digitalmente il proprio applicativo prima dell'invio ad Apple è una condizione necessaria perché questo venga approvato; la procedura è strutturata nella sequenza di tre semplici passi:

1. Creazione del *Certificato di Distribuzione*
2. Creazione di un *App ID* di tipo generico (*) o specifico (ad esempio: *com.website.appname*)
3. Creazione di un *Provisioning Profile di Distribuzione* per App Store.

Prima di tutto è una buona abitudine chiudere completamente la suite di XCode per evitare problemi di mancata sincronizzazione con i certificati che andremo ad installare: questo è uno degli errori più frequenti che spesso bloccano proprio nelle fasi finali uno sviluppatore alle prime armi. Il primo passo, quello di creazione del certificato, richiede che si invii ad Apple un file identificato nella documentazione come *Certificate Signing Request*. Per creare tale documento è sufficiente effettuare alcuni passi all'interno dell'applicazione chiamata *Keychain Access* ("*Accesso Portachiavi*" nella versione italiana



Fig. 1: Le diverse fasi necessarie a seconda del testing o della pubblicazione/ad-hoc distribution

di Mac OS X) installato nella sottocartella *Utility* delle Applicazioni. Dopo l'avvio dell'applicativo effettuiamo la richiesta di un certificato verso una CA (*Certification Authority*).

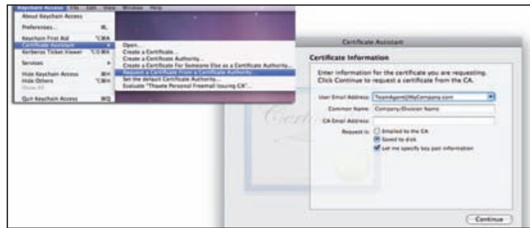


Fig. 2: Le due schermate della richiesta del certificato

Nel campo di testo dedicato all'email è necessario inserire quella utilizzata durante la fase di registrazione come sviluppatore iPhone, anche nel caso di quello dedicato al nome del richiedente deve combaciare con le credenziali inserite durante la creazione del profilo. Il campo *CA Email Address* deve essere lasciato vuoto. Si dovrà poi selezionare il radio button *"Save to disk"* e il checkbox *"Let me specify key pair information"* per farlo sul proprio computer allo scopo di inviarlo successivamente attraverso il Provisioning Portal. La dimensione della chiave deve essere di 2048 bits e l'algoritmo RSA.

PROVISIONING PORTAL

Abbiamo ora il file che ci consentirà di richiede la firma necessaria per i nostri applicativi; accediamo all'iPhone Dev Center visitando il seguente indirizzo , http://developer.apple.com/iphone/ per poi seguire il link presente sul lato destro chiamato iPhone Provisioning Portal (link diretto http://developer.apple.com/iphone/manage/overview/index.action). La guida chiamata *iPhone Developer Program User Guide* è la fonte migliore per chiarire qualunque dubbio abbiate, ed è scaricabile dal link presente nel lato destro della homepage del Provisioning Portal. Entriamo nella sezione chiamata *Certificates*, selezioniamo il tab posizionato in alto nella pagina chiamato *"Distribution"* e clicchiamo

Fig. 4: In questa schermata si inseriscono tutte le informazioni relative all'utente che intende acquistare la licenza

sul pulsante *"Add Certificate"*. Inviamo infine il file che abbiamo creato adoperando il form. Dopo un periodo di tempo che può variare da alcuni secondi a diversi minuti a seconda del carico dei server Apple (durante il quale lo stato del certificato è *"pending"*), troveremo all'interno della pagina il certificato con lo stato di *"issued"* (basterà aggiornare la pagina per verificare l'approvazione) e associato ad una data di scadenza. Clicchiamo sul tasto *download*. A seconda del browser adoperato, il certificato verrà installato automaticamente all'interno di *Keychain Access*, oppure si dovrà fare doppio click sul file. È importante salvare una copia di backup di questo certificato perché, in caso di formattazioni o crash del computer, lo si potrebbe perdere. Ora dobbiamo creare un *App ID* di tipo generale, identificato con il simbolo di asterisco: per fare ciò rechiamoci nella sezione *App IDs* e selezioniamo il bottone presente in alto nella pagina chiamato *"New App ID"*. Ci troviamo ora all'interno di una pagina contenente tre campi: nel primo inseriremo la descrizione del certificato, questa ci consentirà di identificare velocemente all'interno del *Portal* tale identificativo; impostiamo il campo *Bundle Seed ID* al valore *"Generate New"*, nel caso non lo fosse già; l'ultimo campo, *Bundle Identifier*, deve contenere il carattere *. Apriamo una parentesi su quest'ultima configurazione: adoperando l'asterisco, si crea un app id generico che può essere utilizzato, in sinergia con il certificato e il provisioning profile, per firmare qualunque applicazione ma inibisce dalla possibilità di adoperare le *Apple Push Notification*, l'*In App Purchase*, o il *Game Center*, per tale motivo se una o più di queste funzionalità fosse richiesta sarà obbligatorio creare un App ID specifico per la singola applicazione, adoperando uno o più termini che identifichino univocamente l'app. Apple consiglia in questi casi di adoperare il reverse domain name-style: una stringa contenente una serie di valor i seguiti da punto. Esempi di reverse domain potrebbero essere *cognome.nome.nomeapplicazione*, oppure *it.sitoweb.nomeapplicazione*. Torniamo alla creazione dell'App ID generico, al termine della creazione troveremo nell'elenco il nostro App ID identificato da una sequenza di dieci caratteri seguiti da un punto e un asterisco (*xxxxxxxxxx.**); la stringa casuale viene generata automaticamente.

Entriamo nella sezione chiamata *Provisioning*, selezioniamo il tab in alto chiamato *Distribution* e clicchiamo sul tasto *New Profile*. Selezioniamo quindi *App Store* quale distribution method, associamo un nome a tale profilo, e selezioniamo come App ID quello creato adoperando l'asterisco. Confermiamo, e ricarichiamo la pagina finché non risulterà nello stato di *Active*. Ogni riga della pagina contente i profili presenta il nome e l'APP ID associato, che questo caso sarà uguale al nostro *xxxxxxxxxx.**. Scarichiamo infine il profilo appena creato e effettuiamo doppio clic per installarlo nel computer. Sia il certificato che il profilo di distribuzione hanno un termine di scadenza, dopo il quale è necessario effettuarne il rinnovo, operazione che consiste semplicemente nel premere il tasto chiamato *renew* posizionato alla destra delle





NOTA

INDIRIZZI DA RICORDARE

Per l'acquisto della licenza uesto è necessario collegarsi all'indirizzo: http://developer.apple.com/programs/iphone

È un passo obbligato per diventare sviluppatore iPhone.

Questo l'indirizzo di iTunes Connect: https://itunesconnect.apple.com

Qui trovate l'iPhone Provisioning Portal: http://developer.apple.com/iphone/manage/overview/index.action



rispettive voci. Dopo tale rinnovo è sufficiente installare nuovamente i certificati nel proprio computer per continuare a lavorare normalmente.

I CERTIFICATI NECESSARI PER IL TESTING

Prima di concludere questa fase, che si rivela generalmente la più problematica per molti sviluppatori, trattiamo anche di come utilizzare il Provisiong Portal per creare dei certificati utilizzabili per testare sul proprio dispositivo fisico il software/gioco che avete realizzato. Per installare la versione di sviluppo del proprio software su qualunque iPhone/iPad/iPod Touch è necessario:

1. Creare un certificato per lo sviluppo (tab *Development*, invece di *Distribution*) seguendo gli stessi passi effettuati per creare quello di distribuzione adoperando la stessa Certificate Signing Request creata per la distribuzione).
2. Registrare uno o più dispositivi, inserendo il codice chiamato *UDID*;
3. Creare un provisioning profile per lo sviluppo (tab *Development*, invece di *Distribution*) seguendo gli stessi passi effettuati per creare quello di distribuzione avendo cura di selezionare il certificato, l'App ID e i dispositivi che verranno adoperati.

Quando si creano i certificati di provisioning conviene sempre inserire nel nome un "codice" per indicare se è di sviluppo o di distribuzione, in quest'ultimo caso conviene inserire anche una stringa se è per Apple Store o per la AdHocDistribution. In tal modo non si avranno dubbi su quale selezionare all'interno di XCode. Un serie di Provisioning profile potrebbe essere la seguente:

- *LeganzaDevelopment* (per il testing)
- *LeganzaAdhocDistribrution* (per la distribuzione adhoc)
- *LeganzaAppStoreDistribution* (per la pubblicazione su Apple Store)

Ribadiamo la buona regola di salvare tutti questi file in una o più cartelle, possibilmente su diverse periferiche per avere una maggiore sicurezza in caso di danni hardware. Dopo avere creato e installato un nuovo certificato per lo sviluppo, è necessario inserire almeno un identificativo univoco chiamato UDID (Unique Device ID) associato al dispositivo che si utilizzerà all'interno della sezione chiamata *Devices*: per ottenere questa stringa alfanumerica è sufficiente adoperare il tool chiamato *Organizer*, fornito con XCode (*Window->Organizer*), selezionare il dispositivo collegato nell'elenco presente nella colonna sinistra e copiare (è un testo selezionabile) la stringa associata alla voce "*Identifier*". Tramite iTunes (sia Mac OS che Windows) basta selezionare il dispositivo collegato e, dopo aver cliccato sul campo *Identifier* presente sotto la stringa *Software Version* premere la sequenza *"mela"+ C (CTRL+C)*, Questa seconda modalità è utile nel caso si desideri effettuare testing su un dispositivo di un altro utente perché consente di ottenere l'UDID del destinatario senza richiedere che installi la suite XCode. Nel caso comparisse invece di "*Identifier*" la voce "*Serial Number*" basterà cliccare una o più volte su tale riga per visualizzare l'identifier. Una terza soluzione per ottenere un UDID consiste nell'installare una tra le numerose applicazioni gratuite disponibili su Apple Store che consentono di inviare tale codice ad un indirizzo di posta elettronica.

Inseriamo un nuovo dispositivo cliccando su "*Add Devices*" compilando il primo campo con un nome a nostra scelta che identifichi l'hardware adoperato (*Nomeutente iPhone, Nomeutente iPad* etc), e il secondo con l'UDID appena copiato.

Dopo aver inserito uno o più dispositivi è necessario creare un nuovo provisioning profile di tipo *Development* e selezionare i dispositivi associati: è possibile aggiungere o rimuovere successivamente quelli autorizzati, ovviamente provvedendo poi a sostituire il vecchio provisioning profile installato nel proprio computer con la nuova versione aggiornata. Solo all'interno del Provisiong Portal è possibile conoscere in dettaglio quali dispositivi sono abilitati perché all'interno del tool Organizer verrà visualizzato solo un valore numerico pari a quanti sono configurati nel profilo. Nel caso lo desideriate, è possibile effettuare la *Ad-Hoc distribution*, la quale consente di distribuire ad altri utenti il vostro applicativo la versione finale, senza passare per Apple Store. In pratica si firma un'applicazione con le stesse procedure necessarie per inviarlo allo Store, utilizzando però un profilo di provisiong diverso. Si invierà il software insieme a tale profilo all'utente che potrà così testarlo. I passi sono i seguenti:

1. Creare un Provisiong Profile di tipo *Distribution*, selezionando la voce "*Ad Hoc*", utilizzando come certificato quello di *Distribuzione* (lo stesso per pubblicare su Apple Store), e abilitando i dispositivi su cui verrà installato;
2. Firmare la vostra applicazione adoperando questo certificato;



Fig. 3: La posizione dell'UDID in iTunes (sopra) e Organizer (sotto)



PUNTO DI PARTENZA

Un porto sicuro per iniziare l'esplorazione dello sviluppo iPhone: http://developer.apple.com/iphone/ Qui potete creare il vostro account, scaricare l'SDK e consultare la documentazione previa registrazione gratuita.

3. Inviare Provisioning Profile e l'applicazione all'utente via e-mail/link web;
4. Installare entrambi i file all'interno di iTunes trascinandoli nell'area delle applicazioni scaricate;
5. Sincronizzare il dispositivo.

Il limite della Ad Hoc Distribution consiste nel fatto che ha una durata limitata, non è quindi un'alternativa ma serve per testing o per fare pubblicità distribuendo ad alcuni partner una versione a tempo del software, in genere prima del rilascio. Al momento la durata è sei mesi, dopo i quali è necessario aggiornare il profilo ("*renew*") e inviarlo di nuovo all'utente per farlo installare nel telefono. Siamo finalmente giunti all'ultimo passo da effettuare all'interno del provisioning portal, ora dobbiamo tornare ad XCode per confezionare il nostro applicativo per poi accedere alla sezione *Itunes Connect* per inviarlo ad Apple.

L'UTILITY ORGANIZER

Dopo la creazione e l'installazione dei vari certificati (che siano di testing o di distribuzione) è possibile avviare di nuovo XCode. Organizer è una utility accessibile dal menu *Window* di XCode, e rappresenta uno degli strumenti più utile per effettuare le più comuni operazioni richieste da uno sviluppatore, infatti è possibile:

- Visualizzare i dispositivi collegati
- Ottenere l'UDID dei dispositivi
- Ripristinare il sistema operativo a una versione precedente
- Installare (anche con il trascinamento) o cancellare i profili di sviluppo o distribution Ad-Hoc
- Installare (anche con il trascinamento) o cancellare manualmente i propri applicativi dopo la compilazione di XCode;
- Visualizzare i file temporanei creati dal proprio software
- Visualizzare la console dei messaggi del terminale selezionato (il log del sistema operativo)
- Scaricare i crash delle applicazioni
- Effettuare screenshot del dispositvo

Appena collegato un dispositivo, e avendo accettando la richiesta di abilitazione allo sviluppo potremo trascinare nel campo *Provisioning* i vari profili (di sviluppo o ad-hoc) e subito dopo trascinare l'applicazione, ovviamente associata a tali profili in fase di compilazione all'interno di XCode.

LA CONFIGURAZIONE PER LA PUBBLICAZIONE

Creiamo un nuovo file dal menu *File-> New File*, selezioniamo la voce *Code Signing* e sulla destra la voce *Entitlements*, il nome del file sarà *Entitlments.plist*, nelle versioni precedenti di XCode, c'era solo un valore (*get-task-allow* da impostare manualmente a *false*), ora è un file più complesso, richiesto in fase di invio ad Apple. Se non è stato fatto in precedenza, è necessario creare un file di tipo *PNG* che verrà utilizzato come icona dell'applizazione, a seconda del target (iPhone e iPod Touch <=3G, iPhone 4, iPad) potrebbe essere necessario creare più versioni con diverse dimensioni. Impostiamo il nome del file di icona all'interno del file *nomeprogetto.plist* creato da XCode, e cambiamo il Bundle identifier inserendo prima di *.${PRODUCT_NAME:rfc1034identifier}* una stringa identificativa (ad esempio *it.leganza*).



A questo punto è necessario creare una configurazione all'interno delle proprietà del progetto, che chiameremo *Production*. Entriamo nelle proprietà del progetto, nel tab *Configurations,* e selezioniamo quella chiamata *Release*, e clicchiamo sul tasto *Duplicate*. Chiudiamo questa finestra e all'interno della colonna di sinistra di XCode scendiamo fino a trovare la cartella *Targets* identificata da un un bersaglio rosso e bianco, al cui interno troviamo una voce con il nome del progetto che abbiamo creato.
Facciamo doppio clic su tale riga, selezioniamo il tab *Build*, avendo cura di selezionare dal menu a tendina come configurazione *Production*, digitiamo nel il campo *Code Signing Entitlements* il nome del file *Entitlements.plist* e, nel campo *Any iPhone OS Device* scegliamo il provisioning profile che abbiamo creato (*LeganzaAppStoreDistribution* o *LeganzaAdHocDistribution* nel caso dell'Ad-Hoc Distribution).
Chiudiamo questa finestra, verificando che nel menu a tendina *Overview* sia selezionato *Device* e *Production*. Compiliamo (tasto *BUILD*) il progetto. Al termine dell'operazione troveremo il file *nomeprogetto.app* all'interno della cartella *PRODUCTS* della colonna sinistra di XCode; trasciniamo questo file sul desktop, o in una cartella a proprio piacimento, e premendo il destro del mouse selezioniamo la voce *Compress* che

Key	Value
▼ Information Property List	(17 items)
Localization native development re	Italy
Bundle display name	${PRODUCT_NAME}
Status bar is initially hidden	☑
Executable file	${EXECUTABLE_NAME}
Icon file	iPhoneAppIcon.png
▶ Icon files	(2 items)
Bundle identifier	it.newton21.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.6
Application requires iPhone enviror	☑
Main nib file base name	MainWindow_iPhone
Main nib file base name (iPad)	MainWindow_iPad
▶ Supported interface orientations	(1 item)
▶ Supported interface orientations (iF	(4 items)

***Fig. 4:** Un esempio di appname.plist: mostra il nome dell'icona adoperata, il bundle identifier (it.newton21) e la localizzazione in lingua italiana*





Laureato in Ingegneria Informatica, certificato Adobe ACE - Adobe Flex 3 and AIR Certified Expert, EUCIP Core, Sun Certified Programmer for JAVA 6, istruttore di nuoto FIN di 2° Livello, è attualmente impegnato in numerosi progetti multimediali, anche con iPhone e iPad, con alcune società nazionali ed internazionali, è contattabile su neogene@tin.it o su www.leganza.it.

creerà un file con estensione *.zip*. In caso di errori in fase di compilazione (segnalati dall'icona rossa presente nell'angolo in basso della finestra dell'editor) nella maggior parte dei casi saranno dovuti a una discrepanza tra il profilo inserito nel *Target* del progetto e la configurazione del progetto. Per il testing e l'Ad-Hoc Distribution basta semplicemente cambiare provisionig profile prima della compilazione.

ITUNES CONNECT

Prima di accedere ad iTunes connect è consigliabile creare delle immagini perché richieste da Apple:

- uno o più screenshot dell'applicativo (adoperando ad esempio Organizer durante la fase di testing sul dispositivo, oppure un programma di cattura dello schermo se si preferisce il simulatore) facendo attenzione a rispettare le dimensioni richieste. Per iPad la massima è 1024*768 oppure 768/1024, per iPhone <=3GS 320*480 oppure 480*320, per iPhone 4 640*960 oppure 960*640; le dimensioni proposte sono le massime, ma è possibile fornire versioni con dimensioni leggermente minori in cui si omette, se presente, la barra presente in alto che mostra lo stato del dispositivo (Wi-Fi, batteria, stato della rete, ora, etc).
- un'immagine di 512*512 pixels (JPG o TIFF) che verrà adoperata da Apple nella pagina dell'Apple Store associata al vostro software.

Tutte le informazioni che riguardano le dimensioni delle immagini sono inserite all'interno del documento chiamato *iTunes Connect Developer Guide*, un documento di 146 pagine molto esauriente scaricabile in PDF dal sito Apple. Accediamo ad iTunes Connect (https://itunesconnect.apple.com/), rechiamoci nella sezione *Manage Your Applications*, clicchiamo sul tasto *Add New Application*; se non adoperiamo sistemi di criptazione rispondiamo *NO*; se utilizziamo

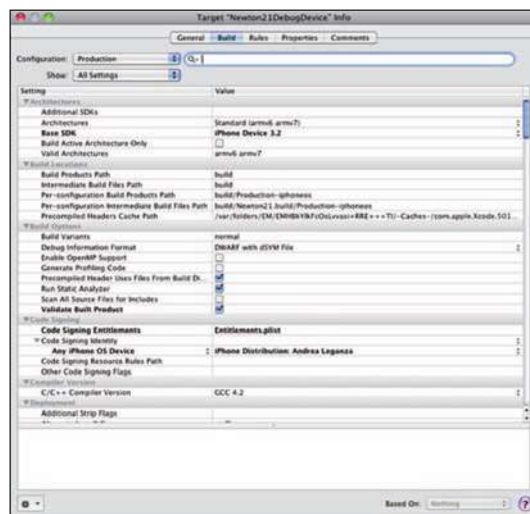


Fig. 5: Come deve apparire la configurazione del Target

la funzionalità di iAds (per SDK dalla versione 4 e successive), rispondiamo *SI* oppure *NO* se vogliamo limitare la visualizzazione dei contenuti, e clicchiamo sul tasto Enable iAds, altrimenti procediamo. Impostiamo in lingua inglese nome del software,

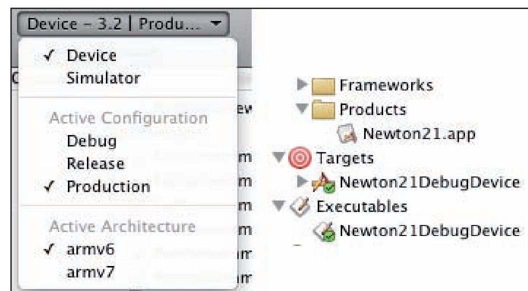


Fig. 6: Sulla sinistra come deve essere impostata la configurazione per la compilazione, sulla destra si vedono l'app creata .app e il target che abbiamo configurato

una breve descrizione. *YES* o *NO* se il software può essere eseguito solo su alcuni dispositivi (un link mostra come configurare il proprio progetto a tale scopo); impostiamo la categoria, copyright, numero di versione (quella impostata nel file *nomeprogetto.plist*, deve variare ad ogni aggiornamento), *sku number* (in genere il nome del software), le parole chiave per consentire che venga trovato quando si effettua una ricerca su iTunes, e le informazioni di supporto (sito web e email). Passiamo alla pagina *RATINGS*, qui si selezionano i campi che identificano il tipo del proprio applicativo (violenza, nudità etc); nella pagina *UPLOAD* si dovrà inviare il file *.zip* del software, l'icona 512*512, e i vari screenshots (selezionate *CHOOSE FILE* per ogni singolo file che volete inserire. Nella localizzazione si aggiungono le informazioni per le diverse lingue, oltre all'inglese, inclusi screenshot. *PRICING*: in questa pagina si può decidere se rilasciare gratuitamente oppure con un prezzo di vendita la propria applicazione, configurare possibili periodi di promozioni per i prezzi, e se limitare la vendita solo in alcuni store. Quasi tutte queste impostazioni sono modificabili anche dopo la pubblicazione. Ora che abbiamo inviato tutte le informazioni ad Apple bisogna attendere per sapere se è stata approvata oppure rifiutata con i suggerimenti del team di valutatori Apple per correggere il problema. Nel caso di pubblicazione di un'applicazione a pagamento (oppure che includa il servizio *iAds*) è necessario registrare preventivamente le proprie coordinate bancarie all'interno della sezione *Contracts, Tax & Banking Information* inserendo il codice del proprio conto corrente, l'IBAN. Per le statistiche dei download queste si trovano nella sezione *Sales and Trends* e *Financial Reports*, oppure basta scaricare sul telefono iTC Mobile App, applicazione gratuita di Apple per iPhone/iPad.

Andrea Leganza

UNA SVEGLIA DIGITALE PER IPHONE

IN QUESTO ARTICOLO MOSTRIAMO COME REALIZZARE UNA SVEGLIA DIGITALE CHE CI AVVISERÀ DI IMPEGNI E SCADENZE IMMINENTI. SARÀ L'OCCASIONE DI APPROFONDIRE I CONCETTI LEGATI ALLA GESTIONE DELL'INTERFACCIA E DEL TIMER







Fig. 1: Al termine del progetto avremo realizzato una sveglia digitale



REQUISITI

Conoscenze richieste
OOP

Software
MacOS X 10.5.4 o superiore, XCode

Impegno

Tempo di realizzazione

In questo articolo mostreremo come realizzare una sveglia digitale. Grazie a questo progetto avremo modo di introdurre alcune classi di estrema utilità: *NSTimer*, fornita dal framework Foundation, che insieme a *UIKit* rappresenta le fondamenta di tutte le applicazioni per iPhone, e *AVAudioPlayer*, presente nel framework *AVFoundation*, il cui scopo è consentire l'esecuzione di brani audio di "qualunque" dimensione. Poiché, come è risaputo, non è permesso nell'iPhone mantenere un'applicazione in background, si potrà utilizzare questa funzionalità di sveglia solo quando la nostra applicazione è effettivamente in esecuzione.

IL TIPO DI PROGETTO

Per realizzare questo software utilizzeremo un nuovo tipo di progetto fornito da *XCode* chiamato *Utility Application*: gli applicativi che vengono identificati con questo termine sono costituiti da due *UIViewController* contenenti una singola *UIView* ciascuno, che si alternano in base alla pressione di un preciso tasto. Il viewcontroller principale viene chiamato automaticamente *MainViewController*.
Il passaggio dall'una a l'altra schermata avviene con la pressione del pulsante *i* nel *MainView*, e con l'utilizzo di quello con label *done*, posizionato in alto a sinistra nella barra di navigazione di *FlipsideViewController*. Il sistema utilizzato per effettuare lo switch è relativamente complesso e non ne parleremo in questo articolo. Essendo la terza serie di articoli prenderemo per scontate tutte le pratiche necessarie per effettuare la creazione degli *IBOutlet* e per prelevare le informazioni da tali oggetti. L'intera business logic verrà implementata all'interno del *MainView Controller*, nel quale abbiamo accesso anche ai contenuti del *FlipsideViewController*.

IL COMPONENTE UIDATEPICKER

Inserendo un componente del tipo *UIDatePicker* all'interno della nostra interfaccia grafica, nel *FlipsideViewController*, forniremo all'utente la possibilità di selezionare una data, con una precisione a nostra discrezione; in questo caso impostiamo, tramite *Interface Builder*, il formato completo (giorno, mese, anno, ore, minuti) settando *mode* al valore "*Date and Time*", la lingua

italiana utilizzando il campo *locale*, e la precisione, con il campo *interval*, ad 1 minuto. *UIDatePicker* fornisce un metodo per ottenere la data selezionata, questa è un'istanza della classe *NSDate* e, quando torneremo al *MainView* provvederemo a memorizzarla in una cartella dell'iPhone, ed utilizzarla per verificare se l'allarme dovrà essere eseguito.

CONTROLLO NSTIMER

Un timer è un sistema che permette di cadenzare l'esecuzione di un metodo con precisi intervalli; ogni timer viene eseguito in un thread distinto. Attenzione, un *NSTimer* non garantisce che l'esecuzione del metodo associato avvenga sempre in un preciso istante, ma che avverrà in un istante pari o successivo a quello desiderato, questa limitazione è dovuta al fatto che, trovandoci in un ambiente multithread e multiprocess, dovremo condividere le risorse hardware con altri processi in esecuzione. In genere ciò avviene utilizzando uno dei cosiddetti "algoritmi di scheduling", i quali provvedono a fornire in maniera ciclica sufficiente tempo di calcolo sulla CPU a tutti i processi che ne facciano richiesta; potrebbe capitare quindi che, quando il nostro timer sia in procinto di scadere, avvenga un evento fuori dal nostro controllo che interrompa o rallenti l'esecuzione del nostro software per alcuni millisecondi, impedendogli di effettuare in tempo l'esecuzione del metodo. La precisione dichiarata è di circa 50/100 millisecondi, ma è comunque non garantita per le motivazioni appena citate.

```
NSTimer *timer = [NSTimer
                    scheduledTimerWithTimeInterval: 1
                                      target: self
                    selector: @selector(handleTimer:)
                                      userInfo: nil
                                      repeats: YES];
- (void) handleTimer: (NSTimer *)
                  timer{ metodo invocato da NSTimer}
```

Con questo codice abbiamo realizzato un timer che viene avviato ogni secondo (o nei millesimi successivi), che allo scadere dell'intervallo esegue il metodo *handleTimer* e che si ripete all'infinito. Il metodo viene immediatamente avviato dopo la sua esecuzione e subisce un retain automatico. Quando, (e se), non avremo più bisogno di questa istanza basterà invocare su di essa il metodo *invalidate*, che provvederà ad effettuare su di essa un release automatico. Per il nostro scopo questo metodo dovrà verificare se è giunto il momento per eseguire l'audio dell'allarme, operazione effettuabile semplicemente confrontando la data attuale, ottenuta richiedendo il valore del campo *date* dalla classe *NSDate*, con quella prelevata dall'*UIDatePicker*:

```
- (void) handleTimer: (NSTimer *) timer {
 if ([alarmDate timeIntervalSinceNow]<=0)
 {
   //viene eseguito il suono dell'allarme
 }
}
```

timeIntervalSinceNow restituisce un valore positivo, i secondi mancanti alla data attuale, quando *alarmDate* è una data futura, mentre i valori sono negativi se *alarmDate* è ormai trascorso.
Abbiamo dovuto utilizzare sia l'uguaglianza che il simbolo di minore uguale perché, come è stato detto, non si può prevedere se il timer verrà eseguito nel preciso minuto in cui dovrebbe scattare l'evento, o in uno dei successivi.

LA MEMORIZZAZIONE DELL'ORA DELL'ALLARME

Dopo aver selezionato una data adoperando l'*UIDatePicker*, provvederemo a memorizzarla in una cartella locale all'iPhone e a caricarla ogni volta che il nostro applicativo verrà eseguito. All'interno del telefono esistono alcune cartelle liberamente accessibili e modificabili nelle quali potremo salvare qualunque tipo di informazione: *tmp* e *Documents*. La prima deve essere utilizzata per creare e gestire informazioni il cui tempo di vita è limitato alla singola esecuzione, mentre la seconda per tutti quei casi in cui un dato deve permanere per diversi avvii dell'applicativo. Esistono svariati metodi per ottenere la corretta posizione di queste cartelle, che sono uniche per ogni software, utilizzeremo il metodo consigliato da Apple quando si realizza un progetto che utilizza la tecnologia *Core Data* (della quale tratteremo in un prossimo articolo) :

```
- (NSString *)applicationDocumentsDirectory {
 return
 [NSSearchPathForDirectoriesInDomains(NSDocument
      Directory, NSUserDomainMask, YES) lastObject];
}
```

Questo codice restituisce il path, una stringa che rappresenta la posizione completa della cartella *Documents*. Questo valore potrà assumere valori diversi a seconda dell'applicativo, ma anche se vi troverete a utilizzare il tutto nel simulatore.
Ii questo caso punterà ad una sottocartella definita in: */Users/$NOMEUTENTE/Library/Application Support/iPhoneSimulator/User/Applications/*, oppu-





***Fig. 2:** Il MainView Controller che mosterà l'allarme*



***Fig. 3:** Il FlipsideView Controller con l'UIDatePicker*



re nel telefono. Per creare un path completo comprensivo del nome del file basterà comporre una stringa utilizzando la seguente procedura:

```
NSString *archivePath = [[self
                applicationDocumentsDirectory]
    stringByAppendingPathComponent:@"sveglia.cfg"];
```

Con questa riga abbiamo realizzato con estrema semplicità un path per un file chiamato *sveglia.cfg* che verrà salvato e caricato quando richiesto. Per verificare se il file sveglia.cfg esiste già all'interno della cartella Documents, basterà semplicemente utilizzare il seguente metodo, *fileExistsAtPath*, fornito dalla classe *NSFileManager*, il quale restituirà *true* in caso affermativo, false in caso negativo:

```
if ([[NSFileManager defaultManager] fileExistsAtPath:
                                        archivePath])
{
  //il file esiste
}
else
{
  //il file non esiste
}
```

Queste informazioni verranno utilizzate per memorizzare la data in cui la sveglia dovrà essere avviata in modo da recuperare e utilizzare tale informazione ad ogni avvio.



NOTA

ULTERIORI APPROFONDIMENTI

Consultare la documentazione online denominata *Exception Programming Topics for Cocoa* per la gestione delle eccezioni e per l'elenco delle eccezioni di default.

AVAUDIOPLAYER

AVFramework, (*Audio Video Foundation Framework*), è un framework che consente di eseguire e anche registrare brani audio *AVAudioPlayer* è una delle classi fornite da tale libreria che consente l'esecuzione di un singolo file audio (uno per istanza); *AVAudioPlayer* consente di mettere in pausa e interrompere un audio, avere informazioni sulla sua durata e sulla posizione in cui è al momento l'esecuzione, consente infine di monitorare i vari livelli di volume assunti dall'audio. Questa classe accetta tutti i formati supportati dall'iPhone:

- AAC
- HE-AAC
- AMR (Adaptive Multi-Rate, a format for speech)
- ALAC (Apple Lossless)
- iLBC (internet Low Bitrate Codec, another format for speech)
- IMA4 (IMA/ADPCM)
- linear PCM (uncompressed)
- µ-law and a-law
- MP3 (MPEG-1 audio layer 3

Il formato suggerito dalla documentazione è 16-bit, little-endian, linear PCM di tipo CAF. È possibile convertire i proprio file audio in questo formato utilizzando il tool chiamato *afconvert* accessibile tramite la finestra di terminale di Mac OS:

```
/usr/bin/afconvert -f caff -d LEI16 {INPUT}{OUTPUT}
```

Nel caso di esecuzione multipla viene consigliato l'utilizzo del formato *IMA/ADPCM* (*IMA4*), mentre per l'ascolto di file singolarmente suggerisce *MP3*, *ALAC* (*Apple Lossless*), *AAC*, *IMA4*. Il primo file che viene eseguito accede direttamente alle risorse hardware, mentre i successivi saranno eseguiti via software. Apple raccomanda nella documentazione di utilizzare questa classe per eseguire qualunque tipo di effetto audio, a meno di avere necessità di gestire in modo distinto i canali stereo, di avere una sincronizzazione precisa, o quando si utilizzano file provenienti da flussi esterni, come avviene ad esempio per le web radio. Apple fornisce numerosi framework oltre ad AVFramework:

- **Media Player framework**: per eseguire brani musicali, audio book, podcasts;
- **Audio Toolbox framework**: per eseguire audio con precise necessità di sincronizzazione, o analisi o conversione, incluso maggiore controllo sulle fasi di registrazione;
- **Audio Unit framework**: per utilizzare plugin audio;
- **OpenAL framework**: viene consigliato come la migliore soluzione per eseguire e gestire musiche per i videogiochi, e utilizza OpenAL 1.1.

Tornando ora a *AVAudioPlayer*, nel nostro progetto utilizzeremo un loop audio, il tipico scandire del tempo di un orologio a tempo, e un suono che avviserà dell'allarme. Prima di effettuare qualunque operazione è necessario aggiungere *AVFoundation.framework* tra i framework che utilizzerà il progetto e importarlo all'interno di *MainViewController* (*#import <AVFoundation/AV Foundation.h>*). Il codice per eseguire un file audio è relativamente breve: prima provvediamo a identificare il path completo della risorsa che ci interessa, (un file mp3 in questo caso), poi creeremo un'istanza di *AVAudioPlayer*, imposteremo il volume, il numero di esecuzioni e lo avvieremo:

```
NSString *path = [[NSBundle mainBundle]
        pathForResource:@"clock" ofType:@"mp3"];
```

```
AVAudioPlayer player = [[AVAudioPlayer alloc]
 initWithContentsOfURL:[NSURL fileURLWithPath:path]
                                          error:nil];
              player.volume = 0.4f;
              [player prepareToPlay];
              [player setNumberOfLoops:-1];
              [player play];
```

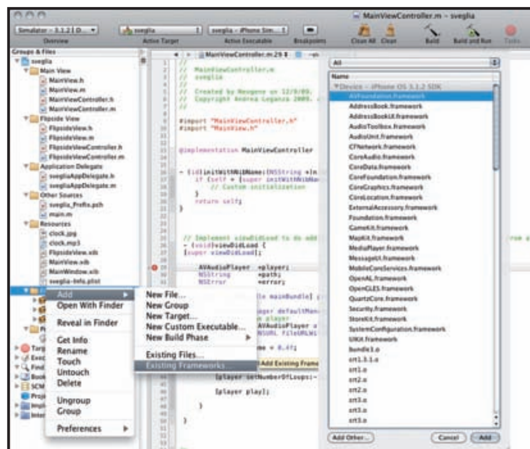


Fig. 1: La procedura necessaria per importare il framework

AVAudioPlayer ha un costruttore che accetta un URL, (che in questo caso sarà la posizione assunta nel nostro software dalla risorsa chiamata *clock.mp3*), e una variabile per memorizzare possibili errori (in questo progetto la abbiamo ignorata, impostandola a *nil*). Il campo *volume* è un *float* il cui intervallo è [*0,1*], dove con *0* si intende il silenzio, mentre con *1* il massimo valore consentito; il metoto *prepareToPlay* memorizza l'audio in un buffer prima di avviare l'esecuzione in modo da evitare interruzioni dovute al caching del file durante il play; *setNumberOfLoops*: può assumere un qualunque valore negativo per indicare un loop infinito, 0 per una singola esecuzione, *i* per (*i+1*) ripetizioni, inserendo 1 si avranno quindi due avvii successivi del suono. Se volessimo monitorare quando un suono è terminato basterà aggiungere il metodo *audioPlayerDid FinishPlaying*: all'interno della nostra classe, appartenente al protocollo *AVAudioPlayerDelegate*:

```
- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer
                    *)player successfully:(BOOL)flag {
        //il suono è  terminato
}
```

GESTITE SITUAZIONI "ECCEZIONALI"

Chi utilizza linguaggi di alto livello, come JAVA o .NET, ha utilizzato generalmente in maniera esaustiva quei costrutti che consentono di catturare uno o più errori generati da una non corretta esecuzione del proprio applicativo.
Un'eccezione è il risultato di un comportamento anomalo, software o hardware, che il programmatore dovrebbe gestire per evitare che il proprio applicativo vada in crash e venga terminato. Per sapere se un metodo o una classe generano una o più eccezioni è necessario consultare la documentazione in linea. La sintassi necessaria per catturare queste eccezioni è estremamente semplice, basta inserire il codice che si vuole monitorare all'interno di un blocco di parentesi graffe a cui si antepone *@try*; con *@catch* si delimita quella parte di codice che dovrà gestire in modo opportuno l'arrivo dell'eccezione, ad esempio deallocando una risorsa, o cercando di risolvere il problema; *@finally* invece è un blocco che viene sempre eseguito e che generalmente viene adoperato per effettuare comuni operazioni di release e pulizia delle risorse utilizzate nel blocco *@try*. Poiché il codice racchiuso da *@finally* viene sempre eseguito risulta estremamente utile perché fornisce un unico punto in cui effettuare le tipiche operazioni di gestione delle risorse, invece di doverle ripetere al termine di *@try* e di *@catch*. Esistono numerosi tipi di eccezioni, tutte istanze di *NSException*, che si distinguono per il nome (il valore del campo *name*, di tipo *NSString*) che queste assumono. Oltre a quelle predefinite, ne esistono anche altre presenti in alcuni precisi contesti, e che sono comunque descritte approfonditamente nella documentazione; incapperete all'inizio molto spesso in *NSRangeException*, quando accederete a indici inesistenti di strutture dati (problema che non si presenta analizzandole utilizzando *Enumerators* e *Fast Enumerators* in ambienti *Thread Safe*), e *NSInvalidArgumentException*, quando passerete parametri non validi ad un metodo.

```
@try {
   //codice da monitorare
}
@catch (tipoeccezione *eccezione) {
    //nel caso avvenga un'eccezione di tipo
                               tipoeccezione viene gestita
}
@finally {
   //questo blocco di codice viene sempre eseguito
}
```

Nel caso di più eccezioni si potranno inserire diversi blocchi *@catch*, in questo caso viene prima analizzato se l'eccezione lanciata dentro *@try* è una versione realizzata ad hoc dal programmatore, di nome *CustomException* e in caso negativo viene confrontata con *NSException*:





RIFERIMENTI WEB
Creazione dell'account, per scaricare l'SDK e consultare la documentazione:
http://developer.apple.com/iphone/



```
@try {
}
@catch (CustomException *ce) {
}
@catch (NSException *ne) {
}
@finally {
}
```

Se si volesse gestire in un unico blocco *@catch* tutte le possibili eccezioni è sufficiente catturare quelle appartenenti alla classe *NSException* che, essendo la più generica da cui derivano tutte le altre, viene sempre riscontrata. Questa pratica è molto comune, ma spesso è più opportuno differenziare i tipi di eccezioni per avere un controllo più specifico di queste situazioni. Realizziamo un semplice blocco di codice in cui teniamo sotto controllo un metodo fornito dalla nostra classe che accetta come parametro un *NSMutableArray* che non ha alcun elemento (è infatti stato inizializzato con capacità nulla);

```
NSMutableArray *anArray = nil;

   array = [[NSMutableArray alloc]
                              initWithCapacity:0];
   @try {
      [self metodo:anArray];
   }
@catch (NSException *exception) {
}
   @finally {
      [anArray release];
   }
```

Nel caso in cui il metodo che abbiamo invocato lanci un qualunque tipo di eccezione siamo in grado di catturarlo e gestirlo come più riteniamo opportuno. Poiché quando si presenta un'eccezione i vari blocchi *@catch* vengono analizzati in sequenza, si procede generalmente inserendo prima quelle eccezioni appartenenti a classi più specifiche, per poi arrivare alle più generiche, dove *NSException* rappresenta la più generica (è anche possibile utilizzare *id* come tipo di eccezione più generale ma non avrete probabilmente mai questa necessità): quando verrà rilevata un'eccezione nel blocco *@try* sarà cura dell'ambiente di esecuzione verificare se il primo *@catch* è adatto, oppure se dovrà procedere con il prossimo. In caso non ne venga trovato almeno uno, tale eccezione verrà inviata all'istanza che contiene quella dove è avvenuto tale evento e, in caso neppure questa sia in grado di gestirlo, continuerà il suo tragitto fino ad un certo punto, identificabile con il contenitore principale *UIApplicationMain* utilizzato all'interno di *main.m*, dopo di che si verificherà un crash del software. Quando viene segnalato nella documentazione in maniera esplicita che l'invocazione di un metodo può lanciare un'eccezione, è sempre consigliato provvedere a gestire tale evenienza. Nel caso non si desideri gestire un'eccezione è possibile rilanciarla, girarla all'oggetto che contiene l'istanza in cui ci troviamo, utilizzando *@throw*:

```
@try {
   //codice che lancia un'eccezione
}
@catch(NSException *e) {
   @throw; // rilancia l'eccezione
}
```

Anche in questo caso verrà utilizzata la procedura di ricerca progressiva di *@catch* in grado di gestirla.
L'utilizzo delle eccezioni dovrebbe essere ristretto solo a questo preciso scopo: gestire comportamenti anomali che richiedono un preciso intervento da parte dello sviluppatore, ma è anche possibile utilizzarle per rappresentare situazioni non anomale, come la pressione di alcune sequenze di tasti, che darebbero quindi inizio ad una precisa sequenza di eventi: tale comportamento, sebbene possibile, viene sconsigliato da Apple.
NSException è corredato di un campo chiamato *userInfo*, un *NSDictionary* nel quale è possibile inserire qualunque tipo di informazione allo scopo di passare alcuni dati al *@catch* che provvederà ad analizzarlo; tale utilizzo verrà spiegato in un altro articolo dove mostreremo come realizzare eccezioni customizzate. Per concludere, può risultare utile mostrare a schermo (in genere viene utilizzato un *UIAlertView*), o in un log, come nel prossimo esempio mostrato, la causa dell'errore e il tipo di eccezione, *NSException*, e conseguentemente tutte le classi derivate, fornisce due campi stringa proprio per questo scopo, *reason* e *name*:

```
@catch(NSException *e) {
   //gestione eccezione
   NSLog (@"EXCEPTION:%@
                        (%@)",e.reason,e.name);
}
```

LA CLASSE NSARCHIVER

La classe *NSArchiver* fornita da Cocoa è in grado di convertire in byte e da byte (in nume-



NOTA

ECCEZIONE

Con il termine "eccezione" si intende un comportamento anomalo del proprio applicativo, generato da cause software o hardware; è sempre consigliato gestire le eccezioni per evitare un crash.

rosi linguaggi viene adoperato il termine *serializzare* e *deserializzare*), istanze di oggetti, scalari, strutture, stringhe e array, ma non consente di effettuare questa operazione per *union*, *void **, puntatori a funzione e sequenze di puntatori. Per quanto riguarda gli *NSArray* e gli *NSDictionary* questi devono contenere solo le strutture dati supportate per essere serializzate/deserializzare. La documentazione è sufficientemente esplicita a riguardo:
Only instances of NSArray, NSDictionary, NSString, NSDate, NSNumber, and NSData (and some of their subclasses) can be serialized. The contents of array and dictionary objects must also contain only objects of these few classes.
Per chi non fosse pratico con tali concetti, queste operazioni permettono di memorizzare lo stato di un'istanza su un determinato file. Adoperando poi la deserializzazione è possibile recuperare questo stato e riutilizzarlo per ripristinare l'istanza nella stessa configurazione precedente. Utilizziamo questa classe per memorizzare all'interno della memoria del telefono la data impostata dall'utente per la sveglia; perché questa procedura funzioni correttamente è necessario che la classe da cui deriva l'istanza su cui utilizziamo *NSArchiver* sia conforme al protocollo chiamato *NSCoding* il quale richiede che vengano implementati i seguenti metodi:

- *(void)encodeWithCoder:(NSCoder *)encoder (formale)*
- *(id)initWithCoder:(NSCoder *)decoder*

Solo *encodeWithCoder* viene esplicitamente richiesto, mentre *initWithCoder*, che provvede alla deserializzazione, è opzionale. Non mostreremo come realizzare il corpo di questi metodi perché la classe che dovremo memorizzare nel telefono, *NSDate*, è già conforme a questo protocollo. Per capire se una classe supporta *out of the box* questa tecnica basterà consultare la documentazione per verificare se *NSCoding* appare tra i protocolli supportati. Questa procedura può essere utilizzata per memorizzare informazioni arbitrarie in maniera estremamente semplice, in un videogioco o in un applicativo, esentando dall'obbligo di creare delle proprie strutture dati, come file xml o di testo, per organizzare tali dati. Per memorizzare la data che l'utente selezionerà adoperando l'*UIDatePicker* basterà utilizzare il metodo della classe *NSKEyedArchiver archive RootObject*: *toFile*: che consente di effettuare la serializzazione immediatamente. Questo metodo restituisce *true* in caso di salvataggio completato, *false* altrimenti.

```
NSString *archivePath = [[self
                    applicationDocumentsDirectory]
   stringByAppendingPathComponent:@"sveglia.cfg"];
NSString storeResult;
if ([NSKeyedArchiver archiveRootObject:alarmDate
                              toFile:archivePath])
{
        storeResult = @"Configurazione salvata con
                                        successo.";
}
else
{
        storeResult = @"Impossibile salvare il file!";
}
```

Come si può vedere, salvare lo stato di un'istanza all'interno di un file locale è un'operazione estremamente semplice; riottenere invece la data all'avvio dell'applicativo comporta l'utilizzo di @try/@catch poichè, come viene esplicitato dalla documentazione: *This method raises an NSInvalidArgument Exception if the file at path does not contain a valid archive.* È possibile evitare di utilizzare questa pratica verificando prima l'esistenza del file utilizzando il metodo *fileExistsAtPath* fornito dalla classe *NSFile Manager*, mostrato in un precedente paragrafo.

```
@try
{
        if ((alarmDate = [NSKeyedUnarchiver
            unarchiveObjectWithFile:archivePath])){
        [alarmDate retain];
        ....
}
@catch (NSException * e)
{
        //Ignoriamo l'eccezione di tipo
   NSInvalidArgumentException che viene lanciata nel
                    caso sveglia.cfg non venga trovato
}
```

CONCLUSIONI

In questo articolo abbiamo introdotto numerosi argomenti interessanti che consentono di realizzare soluzioni anche molto complesse, giochi compresi. Il progetto annesso alla rivista è completo e mostra come si integrano tutti questi concetti. Nei prossimi articoli continueremo questo viaggio nell'universo iPhone e Objective-C. Buona programmazione.

Andrea Leganza





L'AUTORE

Andrea Leganza
Laureato in Ingegneria Informatica, da oltre un decennio realizza soluzioni multimediali, e non, su piattaforme e con linguaggi diversi. Certificato Adobe ACE - Adobe Flex 3 and AIR Certified Expert, e EUCIP Core, appassionato di fotografia, lingua giapponese e istruttore di nuoto FIN, è attualmente impegnato in numerosi progetti multimediali, anche con iPhone, con alcune società nazionali ed internazionali; è contattabile su neogene@tin.it o direttamente sul sito www.leganza.it.

Apple ha aperto una nuova frontiera per la programmazione mobile.
L’avvento di iPhone e di Apple Store ha dato modo agli sviluppatori, anche ai più piccoli, di fare breccia nel mercato mondiale delle applicazioni per dispositivi mobile. Questo approfondimento tematico è pensato per mettere lo sviluppatore in condizione di creare facilmente App per iPhone e pubblicarle su iTunes. La prima parte del testo è una guida passo passo alla creazione di una web application completa per iPhone utilizzando il framework iWebKit.
La seconda parte, invece, è dedicata alla pubblicazione delle App su App Store, con una marcata attenzione al rispetto delle regole per non incorrere nello “stop Jobs” e per favorire le prospettive di business dello sviluppatore mobile. Il testo si chiude con un esempio di progetto pratico che guida il lettore alla creazione di un’applicazione pronta per iTunes.

PuntoInformatico
Internet dal 1996
PuntoInformatico | Libri
www.punto-informatico.it